

# Spring Cloud로 개발하는 マイクロ서비스 アプリケーション



+



Spring  
Cloud

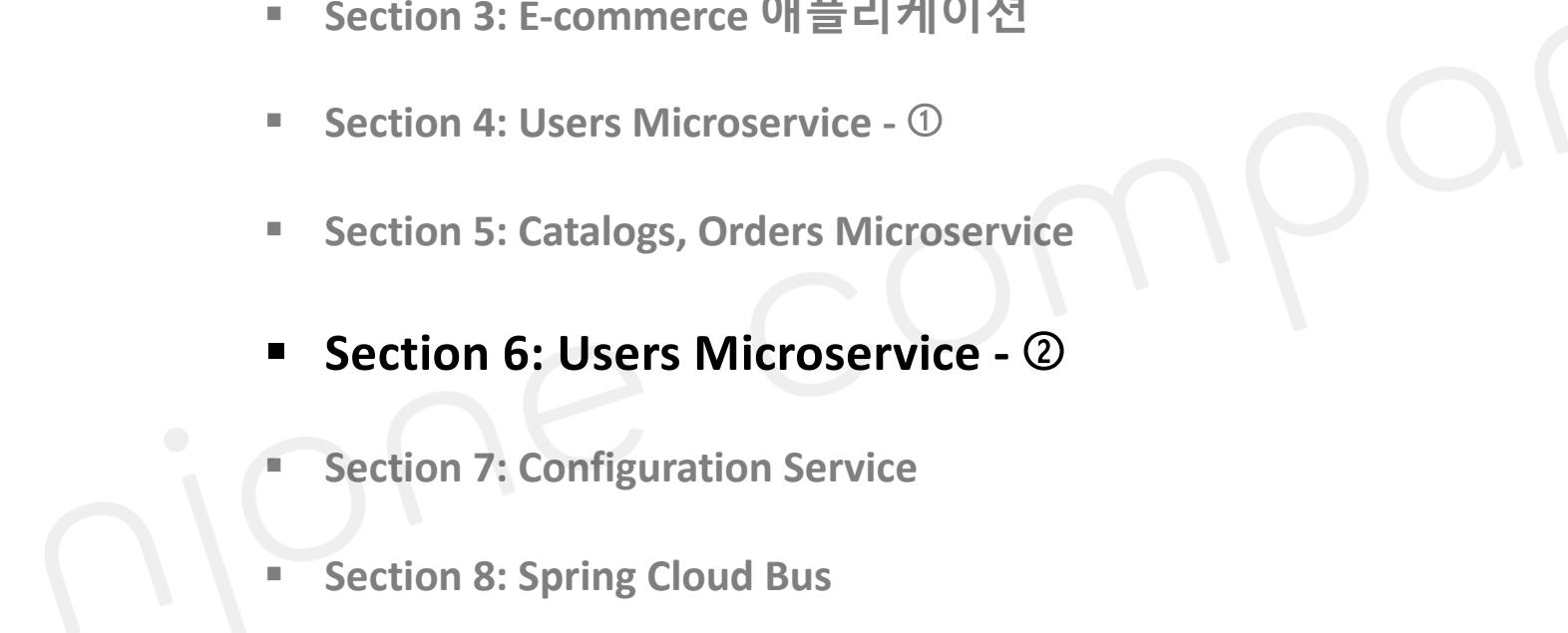
```
CLASS Book
def save(f, title, price, author):
    self.title = title
    self.price = price
    self.author = author
    var fs = require('fs');
    fs.readFile('JSONE.txt' /* 1 */,
    function (err, data) {
        console.log(data); // 3
    });

@Interface NextInnovationDelegate : NSObject < UIApplicationDelegate >
```



## 목차

---

- 
- Part II
- Section 0: Microservice와 Spring Cloud 소개
  - Section 1: Service Discovery
  - Section 2: API Gateway Service
  - Section 3: E-commerce 애플리케이션
  - Section 4: Users Microservice - ①
  - Section 5: Catalogs, Orders Microservice
  - **Section 6: Users Microservice - ②**
  - Section 7: Configuration Service
  - Section 8: Spring Cloud Bus

## Section 6.

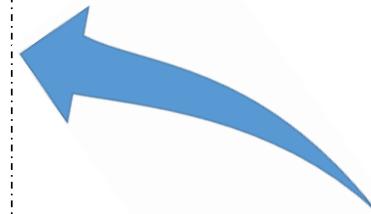
# Users Microservice - ②

- Users Microservice – Login
- JWT (Json Web Token)
- API Gateway service – AuthorizationHeaderFilter

# Users Microservice 기능 추가 – Login

## ■ Features

- 신규 회원 등록
- 회원 로그인
- 상세 정보 확인
- 회원 정보 수정/삭제
- 상품 주문
- 주문 내역 확인



## Users Microservice

Front-end

Business Logic

...  
...  
...  
...



Database





# Users Microservice 기능 추가 – Login

## ■ APIs

기능	URI (API Gateway 사용 시)	URI (API Gateway 미사용 시)	HTTP Method
사용자 로그인	/user-service/login	/login	POST



The screenshot shows a Postman request configuration and its response.

**Request:**

- Method: POST
- URL: `http://127.0.0.1:8000/user-service/login`
- Body tab selected, type: JSON
- Body content:

```
1 {  
2   "email": "test2@test.com",  
3   "password": "12345678"  
4 }
```

**Response:**

- Status: 200 OK
- Headers (9) tab selected
- Body tab selected, showing response data:

KEY	VALUE
token	eyJhbGciOiJIUzUxMiJ9eyJzdWliOiI5NDM1MDFiOS05NDhiLTQ3NzMtOTQ0Ni1kOWI3YTNjI
userId	943501b9-948b-4773-9446-d9b7a3c59b88



# Users Microservice 기능 추가 – Login

- RequestLogin.java

- 사용자 로그인 정보를 저장하기 위한 RequestLogin Model 클래스 추가

```
@Data  
public class RequestLogin {  
    @NotNull(message = "Email cannot be null")  
    @Size(min = 2, message = "Email not be less than two characters")  
    @Email  
    private String email;  
  
    @NotNull(message = "Password cannot be null")  
    @Size(min = 8, message = "Password must be equal or grater than 8 characters and less than 16 characters")  
    private String password;  
}
```



# Users Microservice 기능 추가 – Login

## ■ AuthenticationFilter.java

- Spring Security를 이용한 로그인 요청 발생 시 작업을 처리해 주는 **Custom Filter** 클래스
- UsernamePasswordAuthenticationFilter 상속

- <https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/authentication/UsernamePasswordAuthenticationFilter.html>

org.springframework.security.web.authentication

### Class UsernamePasswordAuthenticationFilter

java.lang.Object

    org.springframework.web.filter.GenericFilterBean

        org.springframework.security.web.authentication.AbstractAuthenticationProcessingFilter

            org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter

- attemptAuthentication(), successfulAuthentication() 함수 구현



# Users Microservice 기능 추가 – Login

- AuthenticationFilter.java

```
public class AuthenticationFilter extends UsernamePasswordAuthenticationFilter {  
    @Override  
    public Authentication attemptAuthentication(HttpServletRequest request,  
                                                HttpServletResponse response)  
        throws AuthenticationException {  
        try {  
            RequestLogin creds = new ObjectMapper().readValue(request.getInputStream(), RequestLogin.class);  
  
            return getAuthenticationManager().authenticate(  
                new UsernamePasswordAuthenticationToken(  
                    creds.getEmail(),  
                    creds.getPassword(),  
                    new ArrayList<>()  
                )  
            );  
        } catch (IOException e) {  
            throw new RuntimeException(e);  
        }  
    }  
  
    @Override  
    protected void successfulAuthentication(HttpServletRequest request,  
                                           HttpServletResponse response,  
                                           FilterChain chain,  
                                           Authentication authResult)  
        throws IOException, ServletException {  
    }  
}
```

# Users Microservice 기능 추가 – Login

## ■ WebSecurity.java

- 사용자 요청에 대해 AuthenticationFilter를 거치도록 설정

```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable();
        // http.authorizeRequests().antMatchers("/users/**").permitAll();
        http.authorizeRequests().antMatchers( ...antPatterns: "/**" ) ExpressionUrlAuthorization
            .hasIpAddress( ipaddressExpression: "192.168.0.8" ) ExpressionUrlAuthorization
            .and() HttpSecurity
            .addFilter(getAuthenticationFilter());
    }

    private AuthenticationFilter getAuthenticationFilter() throws Exception {
        AuthenticationFilter authenticationFilter = new AuthenticationFilter();
        authenticationFilter.setAuthenticationManager(authenticationManager());

        return authenticationFilter;
    }
}
```

A screenshot of a Java code editor showing the `WebSecurity.java` file. The code defines a `configure` method that configures `HttpSecurity`. It includes a block of code that authorizes requests to the `/users/**` path and permits all of them. Below this, there is a more complex configuration for other requests. A red box highlights this complex configuration block. Within this block, a blue box highlights the `hasIpAddress` method call with the value `"192.168.0.8"`. A red arrow points from this highlighted code to the text `Gateway IP`, indicating that requests from this specific IP address will pass through the `getAuthenticationFilter()`.



# Users Microservice 기능 추가 – Login

- WebSecurity.java

- 사용자 요청에 대해 AuthenticationFilter를 거치도록 설정

```
@Configuration
@EnableWebSecurity
public class WebSecurity extends WebSecurityConfigurerAdapter {
    private UserService userService;
    private BCryptPasswordEncoder bCryptPasswordEncoder;
    private Environment env;

    @Autowired
    public WebSecurity(UserService userService, Environment env, BCryptPasswordEncoder bCryptPasswordEncoder) {
        this.userService = userService;
        this.env = env;
        this.bCryptPasswordEncoder = bCryptPasswordEncoder;
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/api/v1/authenticate").permitAll()
                .anyRequest().authenticated()
            .and()
                .formLogin()
                    .loginPage("/login")
                    .usernameParameter("email")
                    .passwordParameter("password")
                    .successHandler(successHandler)
                    .failureHandler(failureHandler)
                .and()
                    .logout()
                        .logoutUrl("/api/v1/logout")
                        .logoutSuccessHandler(logoutSuccessHandler)
                .and()
                    .exceptionHandling()
                        .accessDeniedHandler(accessDeniedHandler);
    }

    private AuthenticationFilter getAuthenticationFilter() throws Exception {
        return new AuthenticationFilter(authenticationManager());
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userService).passwordEncoder(bCryptPasswordEncoder);
    }
}
```

# Users Microservice 기능 추가 – Login

## ■ UserDetailsService 등록

- 기존의 UserService.java 활용

```
public interface UserService extends UserDetailsService {  
    UserDto createUser(UserDto userDto);  
    UserDto getUserByUserId(String userId);  
    Iterable<UserEntity> getUserByAll();
```

```
@Service  
public class UserServiceImpl implements UserService {  
  
    @Override  
    public UserDetails loadUserByUsername(String username)  
        throws UsernameNotFoundException {  
        UserEntity userEntity = userRepository.findByEmail(username);  
  
        if (userEntity == null)  
            throw new UsernameNotFoundException(username);  
  
        return new User(userEntity.getEmail(), userEntity.getEncryptedPwd(),  
            enabled: true, accountNonExpired: true, credentialsNonExpired: true, accountNonLocked: true,  
            new ArrayList<>());  
    }
```

org.springframework.security.core.userdetails

### Interface UserDetailsService

All Known Subinterfaces:

UserDetailsManager

```
public interface UserRepository extends CrudRepository<UserEntity, Long> {  
    UserEntity findByUserId(String userId);
```

# Users Microservice 기능 추가 – Login

## ■ API Gateway Service 설정

- User Service에 대한 Routes 정보 수정

```
routes:  
#  
#  
#  
#  
  - id: user-service  
    uri: lb://USER-SERVICE  
    predicates:  
      - Path=/user-service/**
```



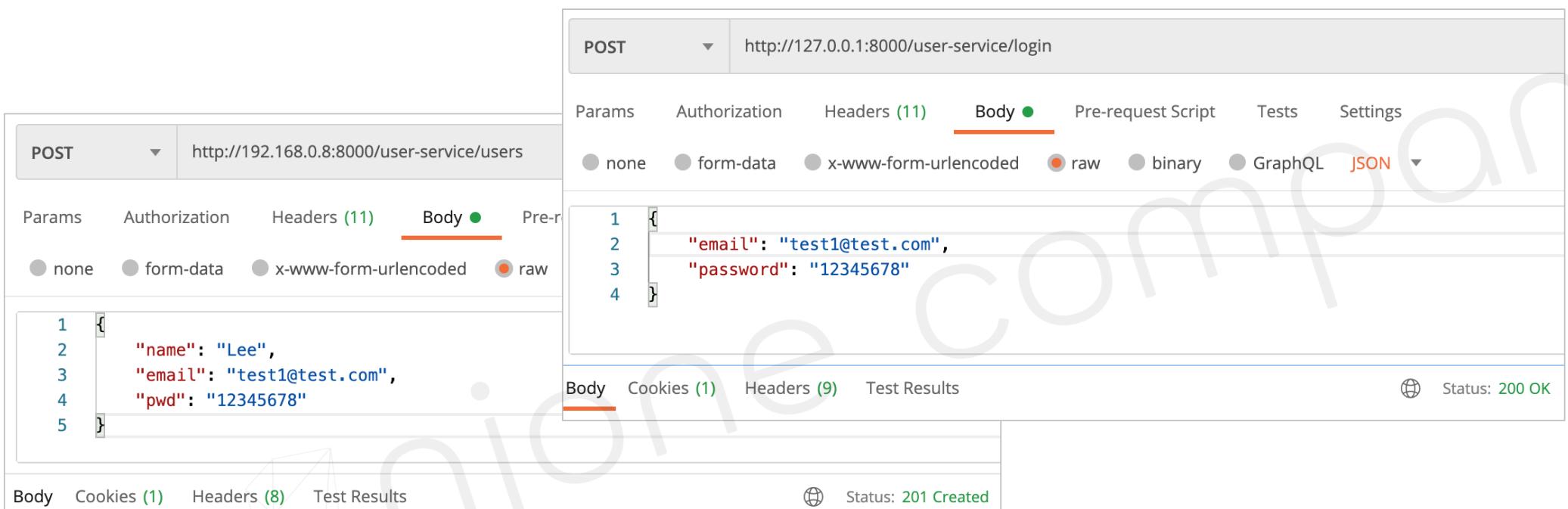
```
- id: user-service  
  uri: lb://USER-SERVICE  
  predicates:  
    - Path=/user-service/login  
    - Method=POST  
  filters:  
    - RemoveRequestHeader=Cookie  
    - RewritePath=/user-service/(?<segment>.*), /$\\{segment}  
  
- id: user-service  
  uri: lb://USER-SERVICE  
  predicates:  
    - Path=/user-service/users  
    - Method=POST  
  filters:  
    - RemoveRequestHeader=Cookie  
    - RewritePath=/user-service/(?<segment>.*), /$\\{segment}  
  
- id: user-service  
  uri: lb://USER-SERVICE  
  predicates:  
    - Path=/user-service/**  
    - Method=GET  
  filters:  
    - RemoveRequestHeader=Cookie  
    - RewritePath=/user-service/(?<segment>.*), /$\\{segment}
```



# Users Microservice 기능 추가 – Login

## ■ Test

- `http://[Gateway IP]:8000/user-service/users` → 회원가입
- `http://[Gateway IP]:8000/user-service/login` → 로그인



The screenshot shows the Postman application interface with two separate requests:

- Request 1 (Left):** POST `http://192.168.0.8:8000/user-service/users`.
  - Body tab selected, showing raw JSON data:

```
1 {  
2   "name": "Lee",  
3   "email": "test1@test.com",  
4   "pwd": "12345678"  
5 }
```
- Request 2 (Right):** POST `http://127.0.0.1:8000/user-service/login`.
  - Headers tab selected, showing 11 headers.
  - Body tab selected, showing raw JSON data:

```
1 {  
2   "email": "test1@test.com",  
3   "password": "12345678"  
4 }
```
  - Status: 201 Created

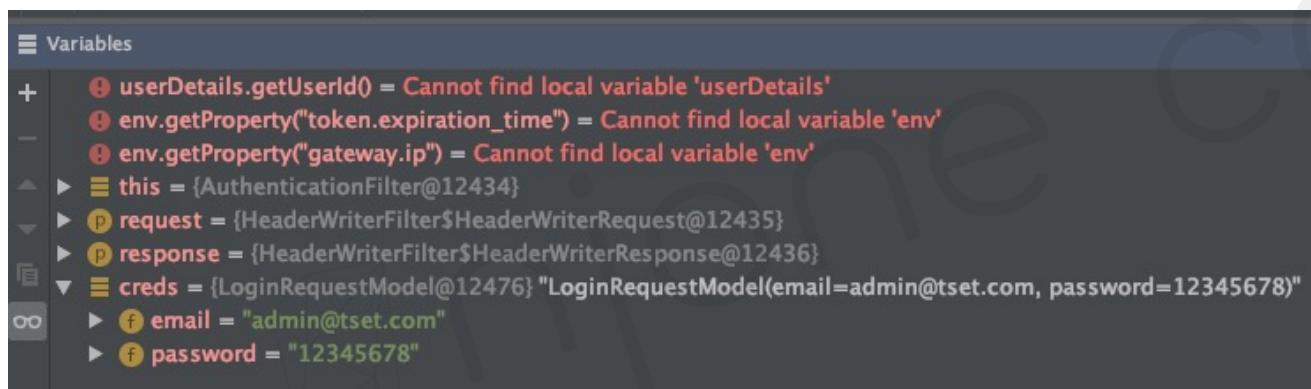


# Users Microservice 기능 추가 – Login

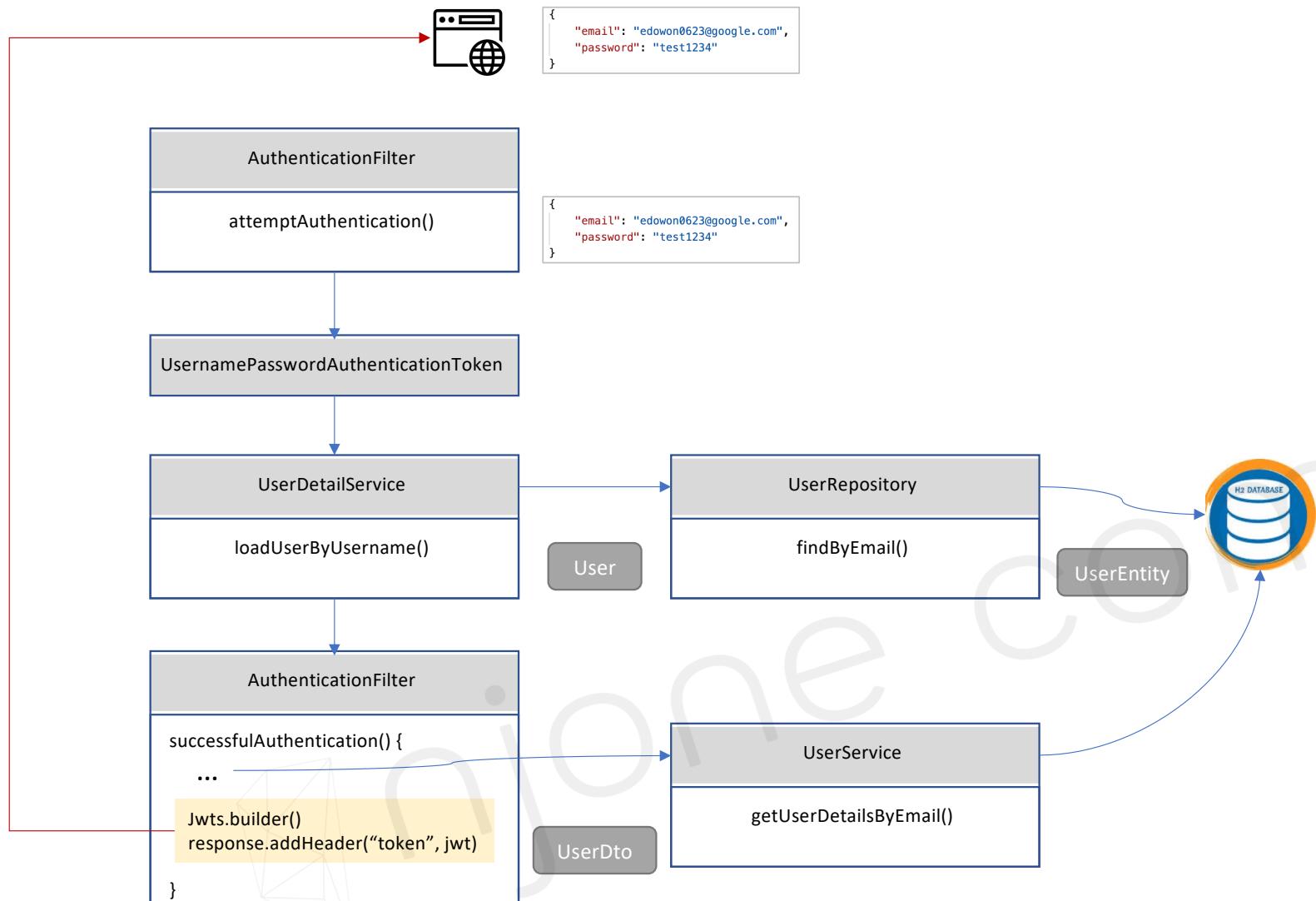
## ■ Test

- Debug

```
42     @Override
43     public Authentication attemptAuthentication(HttpServletRequest request, HttpServletRequest response) throws AuthenticationException {
44         try {
45             LoginRequestModel creds = new ObjectMapper().readValue(request.getInputStream(), LoginRequestModel.class);
46             return getAuthenticationManager().authenticate(
47                 new UsernamePasswordAuthenticationToken(
48                     creds.getEmail(),
49                     creds.getPassword(),
50                     AuthorityUtils.createAuthorityList("ROLE_USER"))
51             );
52         }
53     }
```



# Users Microservice 기능 추가 – Login





# Users Microservice 기능 추가 – Login

- AuthenticationFilter.java의 successfulAuthentication() 수정
  - 인증 성공 시 사용자에게 Token 발생

```
public class AuthenticationFilter extends UsernamePasswordAuthenticationFilter {  
    private UserService userService;  
    private Environment env;  
  
    public AuthenticationFilter(UserService usersService,  
                               Environment env,  
                               AuthenticationManager authenticationManager) {  
        this.env = env;  
        this.userService = usersService;  
        super.setAuthenticationManager(authenticationManager);  
    }  
  
    @Override  
    protected void successfulAuthentication(HttpServletRequest request,  
                                           HttpServletResponse response,  
                                           FilterChain chain,  
                                           Authentication authResult)  
        throws IOException, ServletException {  
        String userName = ((User)authResult.getPrincipal()).getUsername();  
    }  
}
```

AuthenticationFilter.java



# Users Microservice 기능 추가 – Login

- WebSecurity.java의 configure() 설정
  - User Service 사용할 수 있도록 메소드 설정

```
private AuthenticationFilter getAuthenticationFilter() throws Exception {  
    AuthenticationFilter authenticationFilter =  
        new AuthenticationFilter(userService, env, authenticationManager());  
  
    return authenticationFilter;  
}
```

WebSecurity.java

# Users Microservice 기능 추가 – Login

- UserService.java, UserServiceImpl.java, UserRepository.java
  - 사용자 인증을 위한 검색 메소드 추가

```
public interface UserService extends UserDetailsService {  
    UserDto createUser(UserDto userDto);  
    UserDto getUserByUserId(String userId);  
    Iterable<UserEntity> getUserByAll();  
  
    UserDto getUserDetailsByEmail(String email);  
}  
  
@Override  
public UserDto getUserDetailsByEmail(String email) {  
    UserEntity userEntity = userRepository.findByEmail(email);  
  
    if (userEntity == null)  
        throw new UsernameNotFoundException(email);  
  
    return new ModelMapper().map(userEntity, UserDto.class);  
}
```

UserServiceImpl.java

```
public interface UserRepository extends CrudRepository<UserEntity, Long> {  
    UserEntity findByUserId(String userId);  
    UserEntity findByEmail(String email);  
}
```

UserRepository.java

# Users Microservice 기능 추가 – Login

- AuthenticationFilter.java의 successfulAuthentication() 수정
  - 인증 성공 시 사용자에게 Token 발생

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>

token:
  expiration_time: 86400000 #10 days
  secret: user_token
```

```
@Override
protected void successfulAuthentication(HttpServletRequest request,
                                       HttpServletResponse response,
                                       FilterChain chain,
                                       Authentication authResult)
        throws IOException, ServletException {
    String userName = ((User)authResult.getPrincipal()).getUsername();
    UserDto userDetails = userService.getUserDetailsByEmail(userName);

    String token = Jwts.builder()
        .setSubject(userDetails.getUserId())
        .setExpiration(new Date(System.currentTimeMillis()
            + Long.parseLong(env.getProperty("token.expiration_time"))))
        .signWith(SignatureAlgorithm.HS512, env.getProperty("token.secret"))
        .compact();

    response.addHeader("token", token);
    response.addHeader("userId", userDetails.getUserId());}
```

# Users Microservice 기능 추가 – Login

## ■ Test

- `http://[Gateway IP]:8000/user-service/users` → 회원가입
- `http://[Gateway IP]:8000/user-service/login` → 로그인

The image shows two Postman requests side-by-side.

**Left Request (User Registration):**

- Method: POST
- URL: `http://192.168.0.8:8000/user-service/users`
- Body tab selected, showing raw JSON:

```
1 {  
2   "name": "Lee",  
3   "email": "test1@test.com",  
4   "pwd": "12345678"  
5 }
```

**Right Request (Login):**

- Method: POST
- URL: `http://127.0.0.1:8000/user-service/login`
- Body tab selected, showing raw JSON:

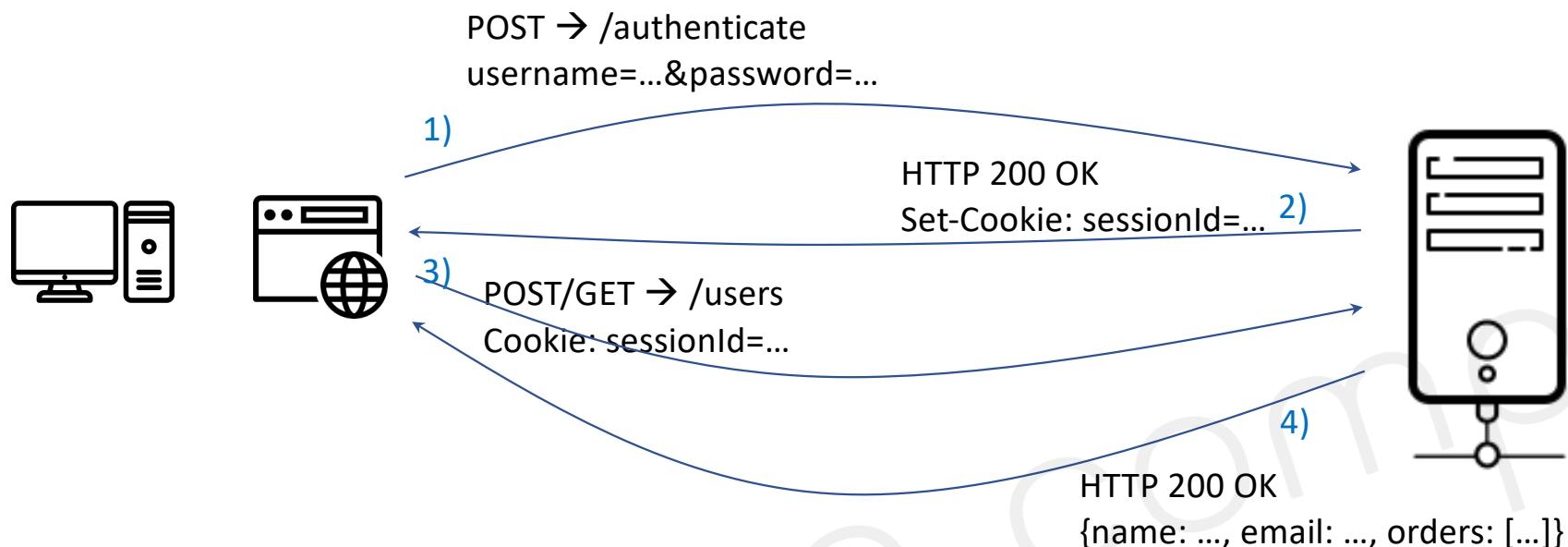
```
1 {  
2   "email": "test1@test.com",  
3   "password": "12345678"  
4 }
```

**Response Headers (from right request):**

KEY	VALUE
token ⓘ	eyJhbGciOiJIUzUxMjMj9.eyJzdWliOjIYTZmMDY4YS0wMTRiLTRiYzUtYWE4Ny01ODEyMDk2
userId ⓘ	ba6f068a-014b-4bc5-aa87-58120963704f

# Users Microservice 기능 추가 – Login

- 전통적인 인증 시스템

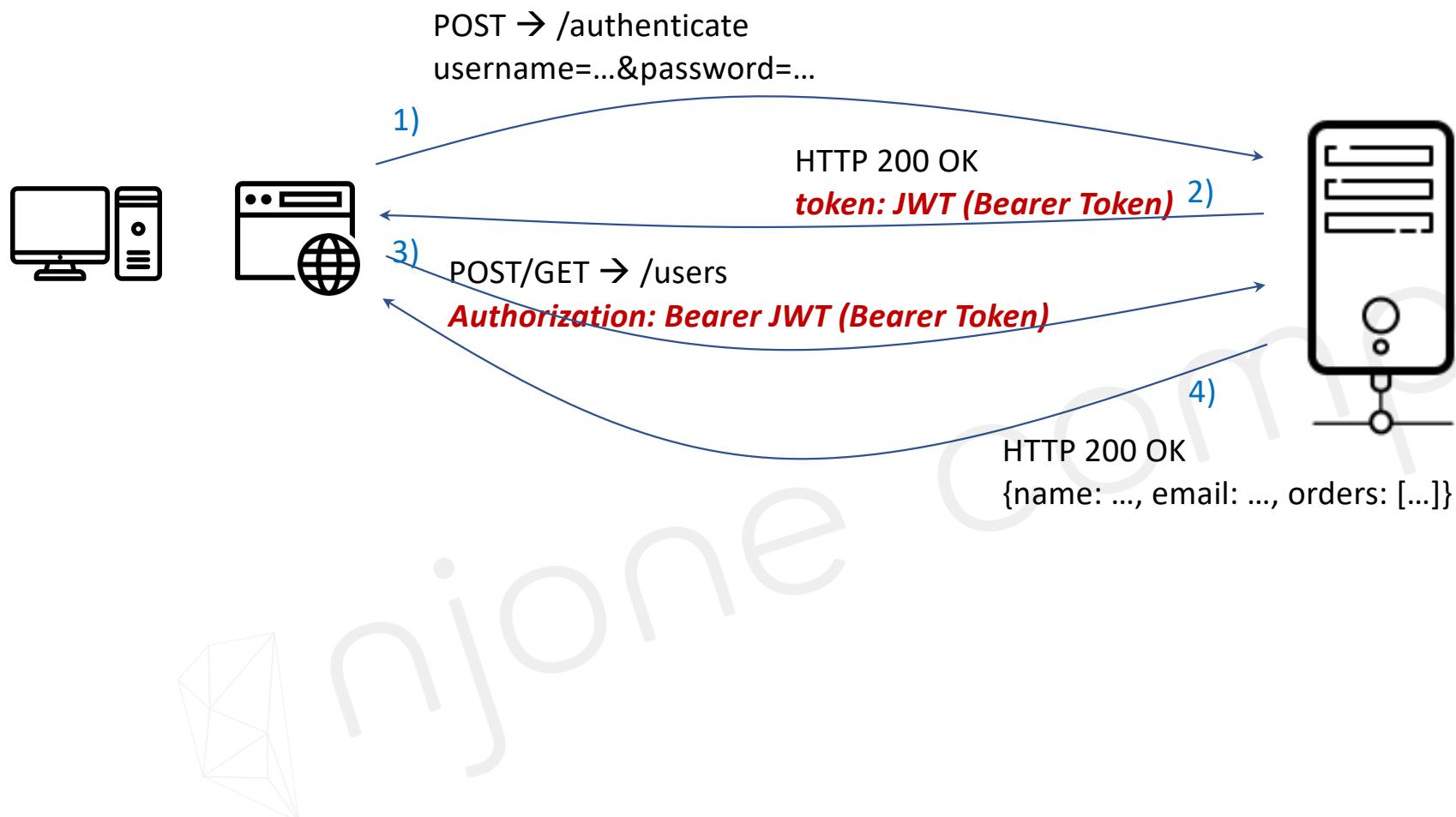


- 문제점
  - 세션과 쿠키는 모바일 애플리케이션에서 유효하게 사용할 수 없음(공유 불가)
  - 렌더링된 HTML 페이지가 반환되지만, 모바일 애플리케이션에서는 JSON(or XML)과 같은 포맷 필요



# Users Microservice 기능 추가 – Login

- Token 기반 인증 시스템

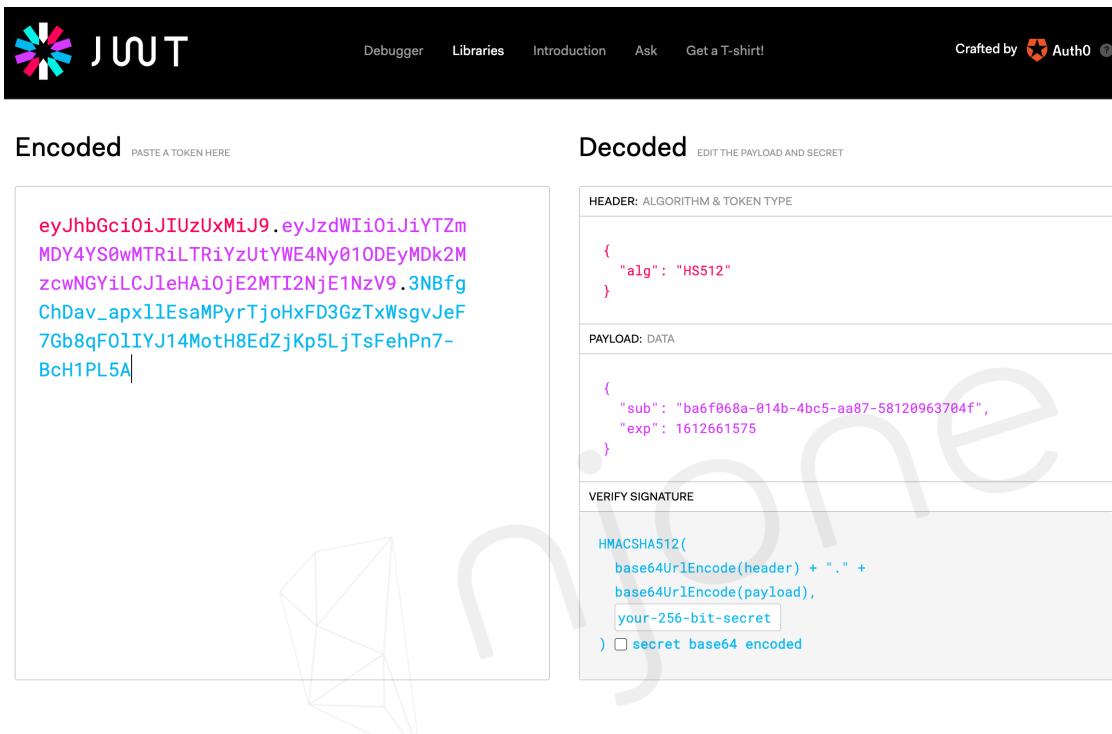




# Users Microservice 기능 추가 – Login

## ■ JWT (JSON Web Token)

- <https://jwt.io/>
- 인증 헤더 내에서 사용되는 토큰 포맷
- 두 개의 시스템끼리 안전한 방법으로 통신 가능



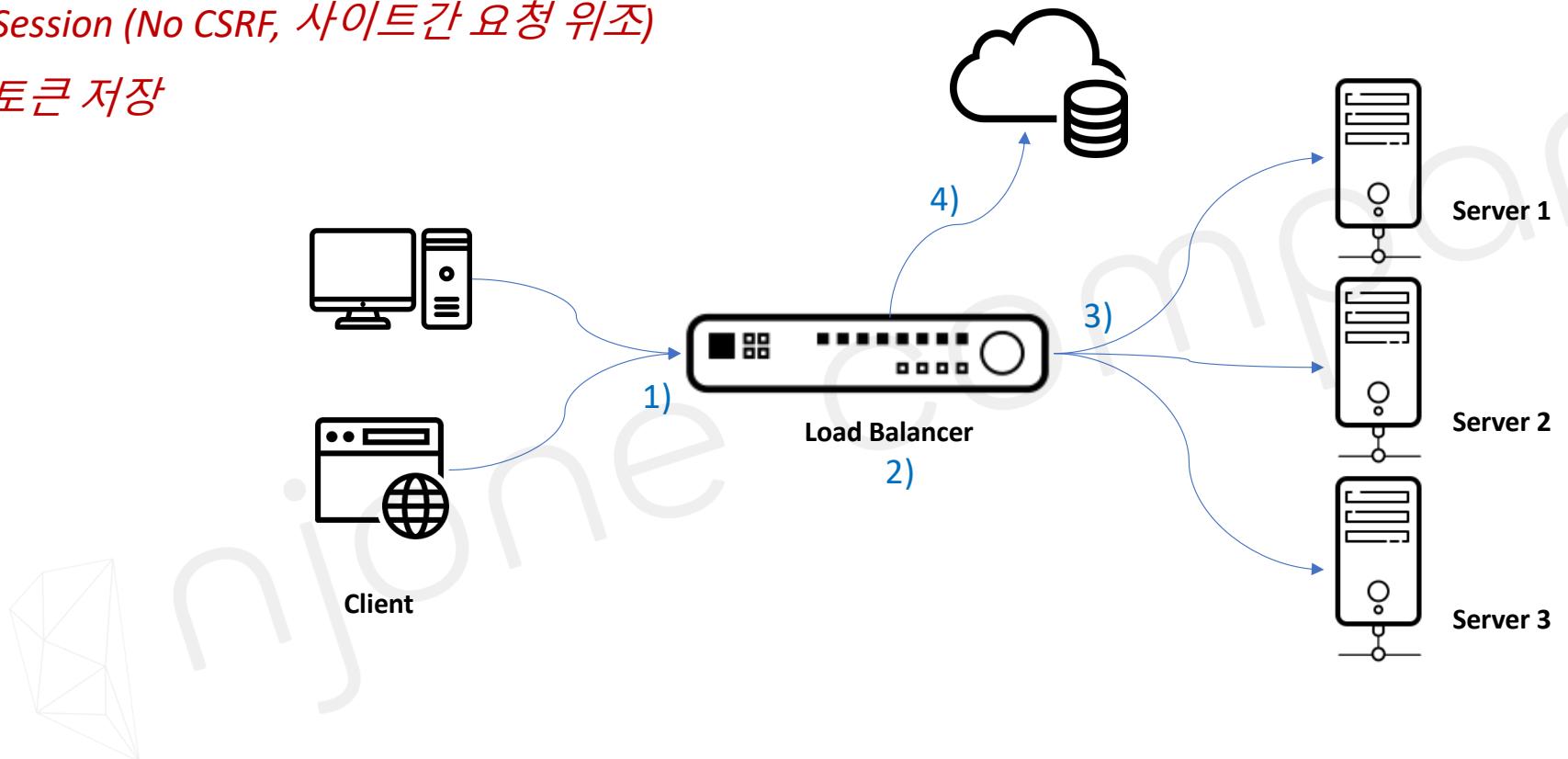
The screenshot shows the JWT.IO decoder interface. On the left, under 'Encoded', there is a long string of characters: eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJiYTZmMDY4YS0wMTRiLTrIYzUtYWE4Ny01ODEyMDk2MzcwNGYiLCJleHAiOjE2MTI2NjE1NzV9.3NBfgChDav\_apxllEsaMPyrTjoHxD3GzTxWsgvJeF7Gb8qF01IYJ14MotH8EdZjKp5LjTsFehPn7-BcH1PL5A|. The right side, under 'Decoded', shows the token structure:  
**HEADER: ALGORITHM & TOKEN TYPE**  
{  
  "alg": "HS512"  
}  
**PAYOUT: DATA**  
{  
  "sub": "ba6f068a-014b-4bc5-aa87-58120963704f",  
  "exp": 1612661575  
}  
**VERIFY SIGNATURE**  
HMACSHA512(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
) □ secret base64 encoded

Language	Library URL
NodeJS	<a href="http://github.com/auth0/node-jsonwebtoken">http://github.com/auth0/node-jsonwebtoken</a>
PHP	<a href="http://github.com/firebase/php-jwt">http://github.com/firebase/php-jwt</a>
Java	<a href="http://github.com/auth0/java-jwt">http://github.com/auth0/java-jwt</a>
Ruby	<a href="http://github.com/progrium/ruby-jwt">http://github.com/progrium/ruby-jwt</a>
.NET	<a href="http://github.com/AzureAD/azure-active-directory-identitymodel-extensions-for-dotnet">http://github.com/AzureAD/azure-active-directory-identitymodel-extensions-for-dotnet</a>
Python	<a href="http://github.com/progrium/pyjwt/">http://github.com/progrium/pyjwt/</a>

# Users Microservice 기능 추가 – Login

- JWT (JSON Web Token) 장점

- 클라이언트 독립적인 서비스 (*stateless*)
- CDN
- No Cookie-Session (No CSRF, 사이트간 요청 위조)
- 지속적인 토큰 저장





# Users Microservice 기능 추가 – Login

- API Gateway service에 Spring Security와 JWT Token 사용 추가
- AuthorizationHeaderFilter.java 추가

```
<!-- jwt token -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```



```
@Component
@Slf4j
public class AuthorizationHeaderFilter extends AbstractGatewayFilterFactory<AuthorizationHeaderFilter.Config> {
    Environment env;

    public AuthorizationHeaderFilter(Environment env) {
        this.env = env;
    }

    public static class Config {...}
```



# Users Microservice 기능 추가 – Login

- AuthorizationHeaderFilter.java

```
@Override
public GatewayFilter apply(Config config) {
    return (exchange, chain) -> {
        ServerHttpRequest request = exchange.getRequest();

        if (!request.getHeaders().containsKey(HttpHeaders.AUTHORIZATION)) {
            return onError(exchange, err: "No authorization header", HttpStatus.UNAUTHORIZED);
        }

        String authorizationHeader = request.getHeaders().get(HttpHeaders.AUTHORIZATION).get(0);
        String jwt = authorizationHeader.replace(target: "Bearer", replacement: "");

        if (!isValid(jwt)) {
            return onError(exchange, err: "JWT token is not valid", HttpStatus.UNAUTHORIZED);
        }

        return chain.filter(exchange);
    };
}
```

# Users Microservice 기능 추가 – Login

## ■ AuthorizationHeaderFilter.java

```
private Mono<Void> onError(ServerWebExchange exchange, String err, HttpStatus httpStatus) {
    ServerHttpResponse response = exchange.getResponse();
    response.setStatusCode(httpStatus);

    log.error(err);
    return response.setComplete();
}
```

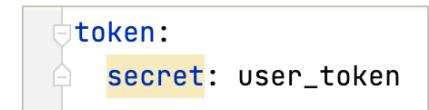
```
private boolean isJwtValid(String jwt) {
    boolean returnValue = true;

    String subject = null;

    try {
        subject = Jwts.parser().setSigningKey(env.getProperty("token.secret"))
            .parseClaimsJws(jwt).getBody()
            .getSubject();
    } catch (Exception ex) {
        returnValue = false;
    }

    if (subject == null || subject.isEmpty()) {
        returnValue = false;
    }

    return returnValue;
}
```





# Users Microservice 기능 추가 – Login

- application.yml

```
- id: user-service
  uri: lb://USER-SERVICE
  predicates:
    - Path=/user-service/**
    - Method=GET
  filters:
    - RemoveRequestHeader=Cookie
    - RewritePath=/user-service/(?<segment>.*), /${segment}
    - AuthorizationHeaderFilter
```

Authenticate 필요 X

Authenticate 필요 O

*POST → /user-service/users (회원가입)*

*POST → /user-service/login (로그인)*

*GET → /user-service/users (회원정보 확인)*

*GET → /user-service/health\_check (상태 체크)*

# Users Microservice 기능 추가 – Login

## ■ Test

- Authorization Type → “No Auth” or “Inherit auth from parent”

POST http://192.168.0.8:8000/user-service/users

Params Authorization Headers (11) Body Pre-request Script Tests Settings

**TYPE**  
Inherit auth from parent

This request is not inheriting any authorization helper at the moment. You can inherit an authorization helper.

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Body Cookies (1) Headers (8) Test Results

POST → /user-service/users (회원가입)

POST → /user-service/login (로그인)

POST http://127.0.0.1:8000/user-service/login

Params Authorization Headers (11) Body Pre-request Script Tests Settings

**TYPE**  
Inherit auth from parent

This request is not inheriting any authorization helper at the moment. You can inherit an authorization helper.

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Body Cookies (1) Headers (9) Test Results

Status: 200 OK

KEY	VALUE
token	eyJhbGciOiJIUzUxMjM9.eyJzdWlvcyI6ImYjMyZS1IMzJmLTRlYzMtODY0YS0xZGM5MmY2NjIwOTIwMDAwMC4wIiwidHlwIjoiSldUIiwiY2lkIjoxLCJpYXQiOjE2NjQwNjUyNjAsImV4cCI6MTY2NjQwNjUyNjAsfQ
userId	b89fb32e-e32f-4ec3-864a-1dc92f63353c

# Users Microservice 기능 추가 – Login

- Test: Authorization Type → “No Auth” or “Inherit auth from parent”

The screenshot shows two separate API requests made using the Postman application.

**Request 1: GET /user-service/users**

- Method: GET
- URL: http://127.0.0.1:8000/user-service/users
- Authorization tab selected: "No Auth"
- Body tab selected: "Pretty" view shows a single item labeled "1".
- Result: Status: 401 Unauthorized

**Request 2: GET /user-service/health\_check**

- Method: GET
- URL: http://192.168.0.8:8000/user-service/health\_check
- Authorization tab selected: "No Auth"
- Body tab selected: "Pretty" view shows a single item labeled "1".
- Result: Status: 401 Unauthorized

Annotations in red text are overlaid on the screenshots:

- GET → /user-service/users (회원정보 확인)
- GET → /user-service/health\_check (상태 체크)

# Users Microservice 기능 추가 – Login

## ■ Test: Authorization Type → “Bearer Token”

### - Bearer Authentication:

- API에 접속하기 위해서는 access token을 API 서버에 제출해서 인증 처리.
- OAuth를 위해서 고안된 방법, [RFC 6750](#)

The screenshot shows a Postman interface for a GET request to `http://127.0.0.1:8000/user-service/users`. The 'Authorization' tab is selected, showing 'Bearer Token' as the type. A warning message about sensitive data is displayed. The 'Body' tab shows a JSON response with user information. The 'Cookies' tab shows the received access token. The status bar at the bottom indicates a 200 OK response.

GET → `/user-service/users` (회원정보 확인)

Bearer Token

eyJhbGciOiJIUzUxMj9.eyJzdWliOjI0DlMyZS1IMzJmLTRlYzMtODY0YS0ZGM5MmY2MzM1M2MiLCJleHAiOjE2MTI2NjU4NjB9.hT2cGvd8E-HrMUi4XjrP-0e9Hwa-cPazrk1Rjksew\_UKkjSs3zJGFEV2Nv\_wSAR72lhby-JNszMvfvHCzxf9Xg

KEY	VALUE
token	eyJhbGciOiJIUzUxMj9.eyJzdWliOjI0DlMyZS1IMzJmLTRlYzMtODY0YS0ZGM5MmY2MzM1M2MiLCJleHAiOjE2MTI2NjU4NjB9.hT2cGvd8E-HrMUi4XjrP-0e9Hwa-cPazrk1Rjksew_UKkjSs3zJGFEV2Nv_wSAR72lhby-JNszMvfvHCzxf9Xg
userId	b89fb32e-e32f-4ec3-864a-1dc92f63353c

로그인 성공 시 전달 받은 Token 정보

# Users Microservice 기능 추가 – Login

## ■ Test: Authorization Type → “Bearer Token”

### - Bearer Authentication:

- API에 접속하기 위해서는 access token을 API 서버에 제출해서 인증 처리.
- OAuth를 위해서 고안된 방법, [RFC 6750](#)

The screenshot shows a Postman interface for a GET request to `http://127.0.0.1:8000/user-service/users`. The 'Authorization' tab is selected, showing 'Bearer Token' as the type. A warning message states: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.' Below it, the token value is displayed as a long string of characters: `eyJhbGciOiJIUzI1NjM5...0e9Hwa-cPazrk1Rjksew_UKkjSs3zJGF...g`. The 'Body' tab is also visible at the bottom.

**GET → /user-service/users (회원정보 확인)**

Body Cookies (1) Headers (8) Test Results

Status: 200 OK Time: 16 ms Size: 364 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "email": "test1@test.com",  
3   "name": "Lee",  
4   "userId": "b89fb32e-e32f-4ec3-864a-1dc92f63353c"  
5 }  
6  
7 ]
```